Sadržaj

1 UVOD

U današnjim je računalima memorija podijeljena na više dijelova koji se razlikuju po količini podataka koju možemo u njih spremiti i po brzini. Najčešće se spominju tri vrste memorije. Hijerarhijski podijeljene od najbrže prema najsporijoj, to su:

- priručna memorija (cache)
- primarna memorija (glavna, radna)
- sekundarna memorija (diskovi)

Razlog podjele je u povezanosti između *brzine i cijene* pojedine memorije. Na primjer, radna memorija omogućuje nekoliko puta brži pristup podacima od sekundarne, ali zbog toga ima skuplju izvedbu. *Veličina* pojedine memorije ovisi njezinoj brzini (sekundarna memorija je najčešće nekoliko stotina puta veća od radne memorije), a samim time i cijeni izvedbe.

Svaka je memorija podijeljena na manje dijelove koji se nazivaju *stranice.* U memoriji se unutar stranica spremaju program, dijelovi programa i podaci koji će procesoru biti potrebni za rad. Veličina pojedine stranice je unaprijed određena i razlikujemo dva osnovna slučaja:

- sve stranice u memoriji su jednake veličine
- u memoriji postoji nekoliko različitih veličina stranica (npr. 512kB, 1MB)

Iako zbog preskupe implementacije nije moguće sve potrebne podatke i programe spremiti u brze memorije (cache, radna memorija), *radna memorija* se prividno u računalima pojavljuje kao memorija koja ima kapacitet sekundarne memorije, a brzinu jednaku brzini najbrže (ili skoro najbrže) memorije u memorijskoj hijerarhiji. To se postiže tako da se stranice koje su potrebne za rad programa prebacuju iz sporije (sekundarne) memorije u bržu (radnu) te procesor nakon toga čita potrebne podatke ili instrukcije iz radne memorije.

Stranica sekundarne memorije koju procesor zahtijeva (koja se kopira u radnu memoriju) nije slučajno odabrana, već podliježe zakonima *prostorne* i *vremenske lokalnosti*:

Vremenska lokalnost govori da će program u bliskoj budućnosti referencirati one programske i podatkovne objekte koje je referencirao i u bližoj prošlosti.

Prostorna lokalnost se očituje u tome što će program u bliskoj budućnosti referencirati one programske i podatkovne objekte koji imaju adrese bliske onima koje su upotrebljavane u bližoj prošlosti.

Zakoni prostorne i vremenske lokalnosti nam omogućuju da u određenoj mjeri predvidimo pojavljivanje budućih zahtjeva za stranicama na temelju zahtjeva generiranih u bliskoj prošlosti. S obzirom da je postupak kopiranja stranica iz sekundarne u radnu memoriju najsporija komponenta u izvršavanju programa, logično je da će (uzevši u obzir zakone prostorne i vremenske lokalnosti) brzina izvođenja programa bitno ovisiti o tome koja će se stranica izbaciti iz radne memorije prilikom zahtjeva za novom stranicom.

U ovom se radu opisuje ideja ostvarivanja što kvalitetnije strategije zamjene stranica upotrebom *genetskog programiranja*. U *drugom* poglavlju rada opisani su osnovni algoritmi zamjene stranica (koje bi bilo poželjno nadmašiti) te neki složeniji algoritmi. U *trećem* poglavlju navode se osnovni pojmovi vezani uz genetsko programiranje i opisuje način rada genetskog programa. U *četvrtom* se poglavlju opisuje ostvarenje strategije izbacivanja stranica iz radnog spremnika genetskim programiranjem i navodi što očekujemo od ostvarenog genetskog programa. U *petom* poglavlju predočeni su rezultati genetskog programa te njihova usporedba s postojećim strategijama izbacivanja stranica.

2 ALGORITMI ZAMJENE STRANICA

Kad iz sekundarne memorije moramo prenijeti novu stranicu u radni spremnik, moramo donijeti odluku koju ćemo stranicu izbaciti iz radnog spremnika. Ta se odluka temelji na ***algoritmu zamjene stranica***. S obzirom da algoritam zamjene stranica izravno utječe na brzinu računalnog sustava, izbor algoritma i njegova izvedba jedno su od ključnih pitanja u organizaciji memorije (*Ribarić*, 1996).

Cilj primjene što djelotvornijeg algoritma zamjene stranica je smanjenje broja ***promašaja***. Promašajem nazivamo zahtjev procesora za stranicom koja se trenutno ne nalazi u radnom spremniku. Nakon svakog promašaja algoritam zamjene stranica određuje koja će se stranica izbaciti iz radnog spremnika te se na njeno mjesto stavlja nova stranica. Ako je procesor generirao zahtjev za stranicom koje se već nalazi u radnom spremniku, radi se o ***pogotku***. Nakon pogotka se sadržaj radne memorije ne mijenja te procesor nastavlja s radom.

Do danas je razvijen velik broj algoritama zamjene stranica, a pokušaji pronalaženja što boljeg algoritma još uvijek traju. Najpoznatiji algoritmi zamjene stranica i njihove karakteristike navedeni su u nastavku.

2.1 Osnovni algoritmi zamjene stranica

OPT (teoretski optimalan algoritam)

Teoretski optimalan algoritam izbacuje iz radnog spremnika stranicu koja se najdalje u budućnosti neće koristiti. Naziv optimalan proizlazi iz činjenice da algoritam ima najmanji mogući broj promašaja.

U praksi taj algoritam nije moguće implementirati jer bismo morali unaprijed znati koje će stranice procesor referencirati. Ipak, algoritam se može koristiti za usporedbu koliko je neka strategija izbacivanja stranica dobra.

FIFO (first-in, first-out)

Kao što se vidi iz imena, FIFO algoritam izbacuje stranicu koje se najdulje nalazi u radnoj memoriji. Iako je algoritam jednostavan i ne zahtijeva nikakve dodatne informacije (osim indeksa stranice koja je najdulje u spremniku), u praktičnim primjenama pokazao se iznimno lošim te se stoga rijetko koristi.

SECOND CHANCE

Second chance je jednostavna modifikacija FIFO algoritma. Svakoj od stranica pridružen je jedan bit koji govori je li se stranica u posljednje vrijeme koristila. Ako je taj bit u nuli, stranica je i stara i nekorištena i izbacuje se. U slučaju da je taj bit u jedinici, bit se postavlja u nulu, stranica se pomiče na kraj reda i postupak se ponavlja.

CLOCK (satni algoritam)

Satni algoritam je modifikacija FIFO algoritma koja ima znatno veću praktičnu primjenu. Algoritam djeluje tako da se sve stranice u radnom spremniku slažu u kružnu listu po redu prispijeća. Prilikom traženja stranice za izbacivanje posebnom se kazaljkom pretražuje lista, počevši od stranice iza zadnje učitane u radnu memoriju. Ukoliko stranica ima oznaku A=1 (stranica se koristila u nekom prethodnom "bliskom" trenutku) ta se oznaka postavlja na 0 te se kazaljka pomiče na sljedeću stranicu. Ako stranica ima oznaku A=0, tada se dotična stranica izbacuje iz radnog spremnika i na isto mjesto se učitava nova stranica. Prilikom učitavanja nove stranice, kazaljka se ne pomiče dalje, već se samo zastavica A postavlja u jedinicu.

NFU (not frequently used)

NFU algoritam iz radnog spremnika izbacuje stranicu koja se u prošlosti *najrjeđe koristila*. Svaka stranica u radnom spremniku ima svoj brojač koji se postavlja u nulu kada se stranica kopira u radni spremnik. Ako procesor referencira stranicu koja se nalazi u radnom spremniku, brojač referencirane stranice se povećava za jedan. U slučaju promašaja, izbacuje se stranica koja ima najmanju vrijednost brojača.

Problem NFU algoritma je taj da pamti broj pojavljivanja svake stranice radnog spremnika bez obzira koliko se daleko u prošlosti ta stranica nije referencirala. Tako će se stranice koje su se bogato referencirale u prošlosti ostati u radnom spremniku bez obzira što te stranice ne moraju biti potrebne za daljnje izvođenje programa (Megiddo, Modha, 2004.).

LRU (least recently used)

LRU strategija iz radnog spremnika izbacuje stranicu koja se najdalje u prošlosti nije koristila. Strategija se temelji na pretpostavci da će se stranice koje su se bogato koristile u bliskoj prošlosti vrlo vjerojatno koristiti i u bliskoj budućnosti. Iako se strategija pokazala vrlo efikasnom, njena implementacija u računalnim sustavima je skupa te se često koriste inačice LRU algoritma (*MRU, ARC*).

Najskuplja implementacija LRU algoritma je pamćenjem svih proteklih zahtjeva za stranicama. Računalo mora održavati i ažurirati vezanu listu što troši znatne vremenske i memorijske resurse.

Nešto bolja implementacija LRU algoritma je pomoću 64-bitnog brojila. Brojilo se inkrementira prilikom svakog referenciranja stranice te se referenciranoj stranici pridružuje vrijednost brojila (početna vrijednost brojila je nula). Ako se referencirana stranica već nalazi u radnom spremniku, vrijednost pridružena toj stranici se osvježava (upisuje se trenutna vrijednost brojila). U slučaju promašaja se iz memorije izbacuje stranica s najmanjom pridruženom vrijednošću, a stranici koja se ubacuje u radni spremnik se pridružuje trenutna vrijednost brojila.

Implementacija LRU algoritma pomoću 64-bitnog brojila se čini jednostavnom, ali nije jednostavno izvediva jer ne postoji adekvatna sklopovska podrška.

NRU (Not Recently Used)

NRU algoritam zamjene stranica daje prednost stranicama koje su se koristile u bliskoj prošlosti. Algoritam za svaku stranicu u radnoj memoriji ima rezervirana dva bita:

- R (read) – postavlja se kad procesor referencira pojedinu stranicu
- M (modified) - postavlja kad se u stranicu nešto upisuje

Prilikom izbacivanja stranice se dijele na četiri skupine (podijeljene prema prioritetu izbacivanja):

1. R=0 i M=0 (nije upotrebljavana, nije modificirana)
2. R=0 i M=1 (nije upotrebljavana, modificirana)
3. R=1 i M=0 (upotrebljavana, nije modificirana)
4. R=1 i M=1 (upotrebljavana, modificirana)

RANDOM

Random algoritam iz radne memorije izbacuje slučajno odabranu stranicu. Algoritam ne zahtijeva pamćenje nikakvih dodatnih informacija, a u praktičnim se primjenama pokazao boljim od FIFO algoritma.

Niti za jedan od gore navedenih algoritama izbacivanja stranica se ne može reći da je bolji od ostalih (ako zanemarimo *teoretski optimalan algoritam*). Omjer broja pogodaka i ukupnog broja zahtjeva za stranicama (još se naziva *hit ratio*) nije konstantan za određenu strategiju, već se mijenja s veličinom radnog spremnika te ovisno o programu koji se trenutno izvodi. Tako, na primjer, u nekim slučajevima LRU strategija može davati najbolje rezultate, dok je za drugi niz zahtjeva bolje u radnoj memoriji čuvati stranice koje su se u prošlosti češće pojavljivale, pa NFU strategija daje bolje rezultate.

2.2 Složeniji algoritmi izbacivanja stranica

Kako bi se poboljšale performanse navedenih strategija, razvijene su strategije izbacivanja stranica koje pokušavaju uzeti u obzir broj zahtjeva za pojedinim stranicama u radnom spremniku te vrijeme zadnjeg pristupa stranici ili se pokušavaju dinamički prilagoditi zahtjevima programa kojeg računalo izvodi. Takve su strategije *ARC te LRU-K (LRU-2).*

LRU-2 strategija izbacivanja stranica pamti dva zadnja referenciranja svake stranice u radnom spremniku te izbacuje stranicu čije je drugo najnovije referenciranje bilo najdalje u prošlosti.

ARC (*Adaptive Replacement Cache*) algoritam izbacivanja stranica pamti koje su se stranice referencirale u bliskoj prošlosti i koje su se stranice najviše referencirale te kombinacijom tih dvaju informacija pokušava povećati broj pogodaka u radnoj memoriji.

Navedene strategije se u većem broju slučajeva pokazuju boljima od osnovnih strategija (LRU, LFU, Clock, FIFO), ali zahtijevaju veće memorijske i vremenske resurse.

3 GENETSKO PROGRAMIRANJE

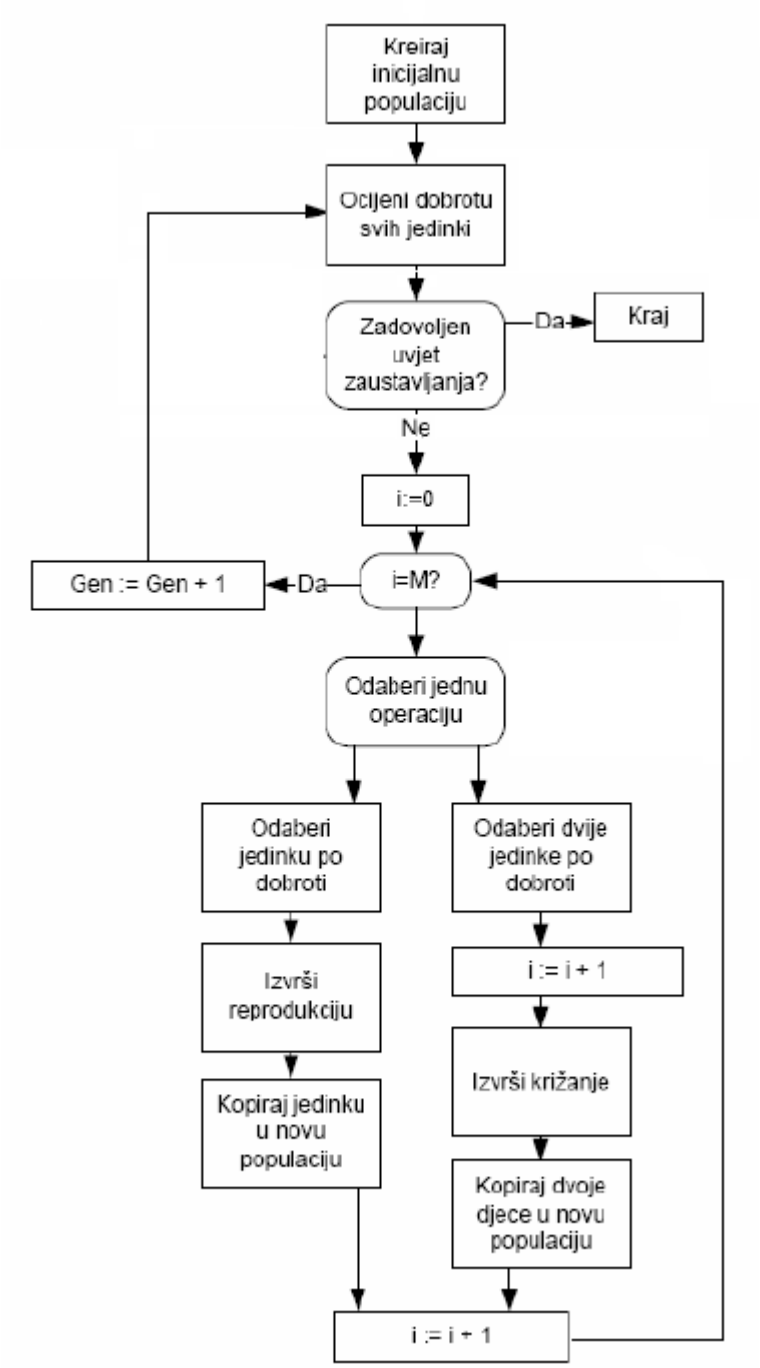
U prirodi snažnije i dominantnije jedinke nadvladavaju slabije te na taj način preživljavaju, prenoseći dobra svojstva svojeg DNA na potomke. Koristeći mehanizme evolucije (prirodna selekcija, križanje te mutacija), priroda stvara jače i prilagodljivije jedinke, dok lošije jedinke (i njihov DNA) izumiru. Jedinke koje su po nekom svojstvu dominantnije od ostalih će daljnjim križanjem proizvesti nove sa svojstvima bližim „savršenstvu“.

Sličan postupak evolucije postoji i u modernom računarstvu, pri čemu je svaka jedinka jedan računalni program koji (bolje ili lošije) rješava zadani problem.

Na slici 3.1. prikazan je osnovni princip optimizacije rješenja (programa) pomoću genetskog programiranja. Najprije se stvara početni skup jedinki (populacija) koji najčešće vrlo loše rješava zadani problem. Nakon toga započinje iterativni postupak određivanja dobrota postojećih jedinki te stvaranja novih postupcima križanja i mutacije (na slici 3.1. *M* predstavlja broj novih jedinki koje stvaramo u jednoj generaciji). Program završava nakon što se zadovolji uvjet zaustavljanja genetskog programa.

3.1 Prikaz jedinki

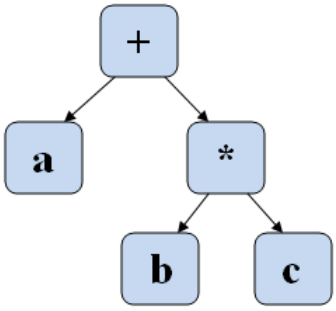
Odabir dobrog načina prikaza jedinki je jedan od najvažnijih preduvjeta da bi neki genetski program pronašao „dovoljno dobro“ rješenje. Jedinke se najčešće prikazuju u obliku stabla, pri čemu svaki čvor predstavlja neko svojstvo jedinke ili izvršava neku funkciju. Svaka jedinka je računalni program koji (bolje ili lošije) rješava zadani problem pa genetski zapis jedinke (čvorovi stabla) mora biti takav da prilikom križanja i mutacije ne dolazi do nemogućih rješenja. Drugim riječima, unutar „definicije jedinke“ moraju biti samo najvažnija svojstva.



Slika 3.1. Optimizacija pomoću genetskog programiranja

3.2 Generiranje početne populacije

Početna populacija (nulta generacija) se stvara na temelju slučajnog odabira. Jedinke u početnoj populaciji najčešće vrlo loše rješavaju zadani problem pa se može reći da je njihova uloga generiranje genetskog materijala pomoću kojeg će se kasnije stvoriti bolje jedinke. Čvorovi stabla (svojstva jedinke) se mogu podijeliti na završne i nezavršne pri čemu samo nezavršni čvorovi mogu biti listovi. Ako smo u nekom trenutku generirali nezavršni čvor, nakon njega moramo generirati dovoljan broj završnih čvorova da popunimo stablo. Na slici 3.2. prikazan je primjer jednostavne jedinke koja izvodi funkciju *f = a + b*c.*



Slika 3.2. jedinka koja izvršava funkciju *f = a + b*c*

Jedinka je dobivena tako da smo redom generirali čvorove „+“, „a“, „*“, „b“ i „c“ te ih dodavali u stablo. Čvorovi „+“ i „*“ su nezavršni čvorovi i zahtijevaju dodatne informacije (završne čvorove) za izvođenje. Završni čvorovi u danom primjeru su „a“, „b“ i „c“. Mogli bismo reći da su završni čvorovi genetski materijal jedinke, dok nezavršni čvorovi opisuju kako se taj genetski materijal odabire i kombinira. Prikaz jedinke u obliku stabla je uobičajen u genetskom programiranju jer se putem njega mogu najlakše izvesti operatori križanja i mutacije.

Izgradnja početne populacije se može izvesti na tri načina:

- *grow* metodom
- *full* metodom
- *ramped half-and-half* metodom

Grow metoda – počevši od korijena, slučajno se odabire vrsta čvora koji će se dodati u stablo. Pritom je moguće birati završne ili nezavršne čvorove. Ako je odabran završni čvor, sustav odabire bilo koji završni čvor i dodaje ga u stablo, inače dodaje neki nezavršni čvor. Postupak se rekurzivno ponavlja dok svi listovi stabla nisu završni ili dok se ne dostigne maksimalna dozvoljena dubina stabla.

Full metoda – slična je grow metodi, ali je unaprijed definirano na kojoj se minimalnoj dubini stabla mogu početi pojavljivati završni čvorovi.

Ramped half-and-half – prilikom generiranja jedinki se unaprijed definira samo maksimalna dubina stabla (*md*). Kreiranje jedinki se obavlja na sljedeći način:

- populacija se podijeli na *md-1* dijelova
- polovica svakog dijela generira se metodom *grow*, a druga polovica metodom *full*

3.3 Računanje dobrote jedinki

Dobrota (kvaliteta) pojedine jedinke je mjera kvalitete te jedinke u zadanom prostoru rješenja (jedinka koja je bolje obavila zadani posao ima veću dobrotu). Za neke je probleme jednostavno izračunati dobrotu jedinki (npr. računanje maksimuma funkcije – veća vrijednost funkcije označava veću dobrotu), dok se kod složenijih problema dobrota može izraziti kao, na primjer, vrijeme trajanja projekta ili vrijeme koje je potrebno da se obavi simulacija sustava koji optimiramo.

Dobro definirano računanje dobrote jedinki je jedna od ključnih stvari za uspješnost genetskog programa jer direktno utječe na izbor jedinki koje prelaze u iduću generaciju.

3.4 Selekcija

Selekcija je proces kojim se osigurava prenošenje boljeg genetskog materijala iz generacije u generaciju i stoga direktno utječe na samu kvalitetu genetskog programa. Postupci selekcije se međusobno razlikuju po načinu odabira jedinki koje će se prenijeti u sljedeću generaciju.

Prema *načinu prenošenja genetskog materijala* selekcija se dijeli na:

- ***Generacijske selekcije***: proces odabire najbolje jedinke i od njih kreira novu generaciju
- ***Eliminacijske selekcije***: proces selekcije eliminira najgore jedinke iz populacije

Generacijske selekcije iz populacije odabiru određen broj najboljih jedinki i od njih stvaraju međupopulaciju. Broj jedinki koje prežive selekciju je manji od veličine populacije pa se jedinke nadomještaju kopiranjem već selektiranih. Postupak može rezultirati nekvalitetnim genetskim materijalom jer se iste jedinke višestruko kopiraju te se gubi genetska raznolikost.

Eliminacijske selekcije odstranjuju slučajno odabrane jedinke iz populacije pri čemu lošije jedinke imaju veću vjerojatnost eliminacije. Eliminirane jedinke se nadomještaju križanjem postojećih.

Prema *načinu odabira pojedinih jedinki* iz skupa jedinki na temelju njihove dobrote selekcija može biti:

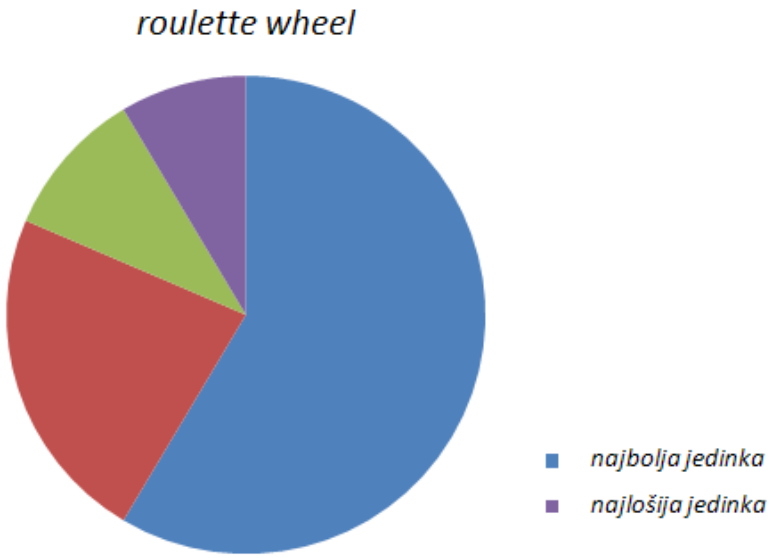
- ***Jednostavna proporcionalna selekcija*** *(roulette-wheel selection)*
- ***K-turnirska selekcija***

Jednostavna proporcionalna selekcija se najjednostavnije može prikazati kao kotač ruleta podijeljen na dijelove različitih veličina (slika 4.3.). Za sve jedinke unutar populacije definira se njihova dobrota te se svakoj jedinki pridružuje dio kotača ruleta proporcionalan njenoj dobroti (podrazumijeva se da u ovom slučaju dobrote jedinki ne

smiju biti negativni brojevi). Nakon toga odabiremo jedinke koje će činiti sljedeću generaciju na sljedeći način:

- slučajno odabiremo jednu jedinku iz populacije, pri čemu jedinke s većom dobrotom imaju veću vjerojatnost odabira
- postupak odabira ponavljamo *N* puta (*N* označava veličinu populacije)

Osim već spomenutog nedostatka da dobrote jedinki ne smiju biti negativni brojevi, glavni nedostatak ovakvog načina odabira jedinki je zagušenje populacije zbog intenzivnog kopiranja dobrih jedinki (program može „zaglaviti“ u lokalnom optimumu).



Slika 3.3. – Roulette-wheel selection

K-turnirska selekcija odabire *k* članova populacije koji sudjeluju u turniru te uspoređuje njihove dobrote (*k* označava veličinu turnira, npr. tri). Jedinke s lošijom dobrotom ispadaju iz generacije te se zamjenjuju novima koje nastaju križanjem jedinki koje su prošle turnirski odabir. Svojstvo takvog načina odabira je ***elitizam***, tj. očuvanje najboljih jedinki u generaciji (trenutno najbolja jedinka se nikad ne će eliminirati).

Algoritam k-turnirskog odabira je sljedeći (Jakobović, 2007):

Početak (GA s k-turnirskim odabirom)

Stvori populaciju P(0)

Ponavljaj

Odaberi k jedinki iz populacije

Pronađi i obriši najlošiju od k odabranih

Generiraj novu jedinku ponoću genetskih operatora

Dok nije zadovoljen uvjet zaustavljanja

Kraj

3.5 Križanje

Križanje jedinki je proces u kojem se rekombinira genetski materijal dvaju roditelja. Rezultat križanja su dvije jedinke koje od svakog roditelja nasljeđuju dio genetskog materijala. Princip križanja (kod jedinki prikazanih u obliku stabla) je sljedeći:

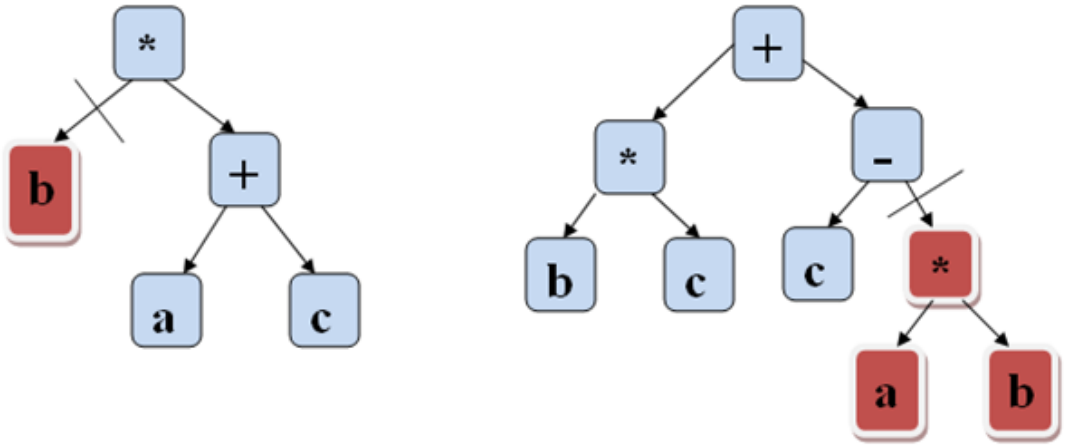
- Slučajno se odabiru točke prekida za oba roditelja (proizvoljna grana stabla)
- Podstabla ispod točke prekida se odvajaju od roditeljskog stabla
- Na praznu vezu koja je nastala micanjem podstabla iz pojedinog roditelja stavlja se podstablo koje je odvojeno od drugog roditelja

Roditeljske jedinke prikazane na slici 3.4. izvršavaju funkcije:

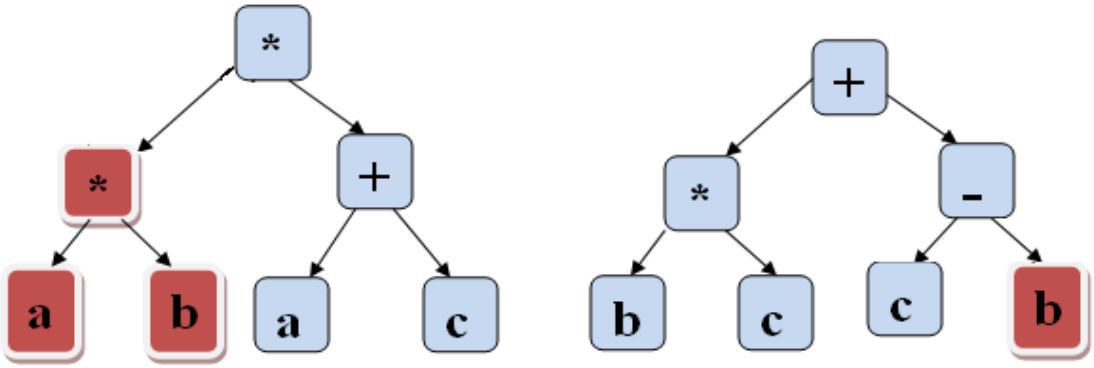
$$f = b * (a + c) \tag{3.1}$$

$$g = (b * c) + (c – (a * b)) \tag{3.2}$$

Križenjem roditeljskih jedinki (podstabla koja se izmjenjuju označena su drugom bojom) dobivene su dvije nove jedinke (Slika 3.5) koje ne moraju nužno biti bolje od svojih roditelja. Ako je novostvorena jedinka bolja od roditeljskih, njezin će se genetski materijal iskoristiti u daljnjoj evoluciji, dok će se jedinke koje su lošije od roditeljskih nakon određenog broja koraka izbaciti iz populacije.



Slika 3.4. Roditeljske jedinke



Slika 3.5. Rezultat križanja

3.6 Mutacija

Mutacija u genetskom programiranju uvodi u nove jedinke neke gene koji će (možda) poboljšati dobrotu jedinke. Jedinke sa „smrtonosnim“ mutacijama će vrlo brzo biti eliminirane u postupku selekcije tako da ne će dugotrajno narušiti prosječnu dobrotu jedinki unutar populacije. Mutacija se u genetskom programiranju izvodi na sljedeći način.

- Iz populacije se odabere jedinka na kojoj želimo izvršiti mutaciju
- U stablu koje reprezentira jedinku se nasumce odabere jedan čvor (ili list)

- Odabrani čvor se zajedno sa svojim podstablom briše iz stabla
- Na tom se mjestu zatim slučajnim odabirom stvara novo podstablo

Osnovna uloga mutacije je izbjegavanje lokalnih optimuma unutar kojih može „zaglaviti“ genetski program te obnavljanje izgubljenog genetskog materijala (koji je možda dobar, ali je bio iskorišten na loš način).

4 OSTVARENJE STRATEGIJE ZAMJENE STRANICA U RADNOM SPREMNIKU GENETSKIM PROGRAMIRANJEM

Genetskim programiranjem se najčešće pokušavaju riješiti problemi čije je rješenje vrlo teško (ili nemoguće) odrediti nekom eksplicitnom formulom ili iscrpnom pretragom. Algoritmi zamjene stranica u radnom spremniku pokušavaju predvidjeti buduće zahtjeve za stranicama na temelju zahtjeva u bližoj prošlosti (npr. LRU) ili pokušavaju pronaći neku pravilnost u pojavljivanju zahtjeva (npr. FIFO). Budući da ne možemo sa sigurnošću reći da je neka strategija bolja od ostalih, a i kvaliteta pojedinih strategija se mijenja ovisno o programu koji se trenutno izvršava na računalu, odabir genetskog programiranja za implementaciju strategije zamjene stranica u radnom spremniku je opravdan.

4.1 Način rada programa i prikaz jedinke

Rad genetskog programa za izbacivanje stranica iz radnog spremnika sastoji se od tri dijela:

- Izbacivanje jedne stranice iz radnog spremnika (izravno ili na temelju dodatnih informacija)
- Upisivanje neke vrijednosti u polje *info1* (informacija koja može poslužiti za odabir stranice koja će se izbaciti)
- Upisivanje neke vrijednosti u polje *info2*

Polja ***info1*** i ***info2*** služe za pamćenje informacija vezanih uz stranice radnog spremnika (informacije mogu, na primjer, biti broj referenciranja pojedine stranice, vrijeme proteklo od zadnjeg referenciranja ili neki slučajan broj). Jedinke genetskog programa te informacije mogu (ali i ne moraju) iskoristiti za odlučivanje koja će se stranica izbaciti. Također, za odluku o izbacivanju mogu se koristiti podaci iz samo jednog ili oba polja informacija (ovisi o strukturi same jedinke). Zapisivanje podataka u polja informacija obavlja sama jedinka prilikom svakog referenciranja stranice sekundarne memorije.

U poljima *info1* i *info2* se u svakom koraku mijenja vrijednost pridružena samo onoj stranici koju je program u prethodnom koraku referencirao (u suprotnom bi algoritam bio iznimno spor).

Jedinke genetskog programa (svaka predstavlja jednu strategiju izbacivanja stranica iz radnog spremnika) prikazane su pomoću ***binarnog stabla*** koje je odgovarajućim čvorovima podijeljeno na tri dijela. Prvi dio stabla odlučuje koja će se stranica izbaciti iz radnog spremnika, dok druga dva upisuju odgovarajuće podatke u polja informacije. Prije detaljnijeg opisa načina rada programa i same strukture jedinke potrebno je definirati čvorove (i listove) pomoću kojih će se *stabla* graditi.

4.1.1 Završni čvorovi

MIN1 - ako se program nalazi u dijelu stabla koji služi za izbacivanje stranice iz radnog spremnika, čvor vraća *indeks* stranice s najmanjom vrijednošću u polju *info1*, dok u dijelu stabla za upis neke vrijednosti u polje informacije vraća *najmanju vrijednost* u polju *info1.*

MIN2 – vraća *indeks* stranice s najmanjom vrijednošću u polju *info2* ili najmanju *vrijednost* u polju u *info2.*

MAX1 – vraća *indeks* stranice s najvećom vrijednošću u polju *info1* ili najveću *vrijednost* u polju *info1.*

MAX2 – vraća *indeks* stranice s najvećom vrijednošću u polju *info2* ili najveću *vrijednost* u polju *info2*.

RANDCS (*rand modulo cash size*) – vraća *slučajan broj* u rasponu od nula do CASH_SIZE-1 (veličina radnog spremnika).

COUNTER – vraća broj koji je na početku jednak nuli te se prilikom svakog zahtjeva za stranicom povećava za jedan (*brojač*).

INCREMENT - ako se program nalazi u dijelu stabla koji služi za izbacivanje stranice iz radnog spremnika, ekvivalentan je čvoru *CMCS*, u suprotnom inkrementira vrijednost koja se nalazi u polju informacije stranice koju je program referencirao u prethodnom koraku.

CMCS (*counter modulo cash size*) – vraća broj koji se dobiva pri ostatku dijeljenja brojača (čvor *counter*) s veličinom radnog spremnika.

CON1, ***CON2*** i ***CON3*** – vraćaju redom konstante jedan, dva i tri.

4.1.2 Nezavršni čvorovi

LR – izvršava redom sva podstabla s lijeva na desno (s obzirom da je genetski program implementiran pomoću binarnog stabla, čvor izvršava najprije lijevo podstablo, a zatim desno).

ADD – vraća zbroj vrijednosti koje dobiva od podstabala.

IF1 – ako je prosječna vrijednost zapisana u *info1* manja od prosječne vrijednosti u *info2*, vraća vrijednost lijevog podstabla (u suprotnom vraća vrijednost desnog podstabla).

IF2 – ako je najmanja vrijednost zapisana u *info1* manja od najmanje vrijednosti u *info2*, vraća vrijednost lijevog podstabla (u suprotnom vraća vrijednost desnog podstabla).

MIN12 – vraća manju između vrijednosti koje dobiva od podstabala.

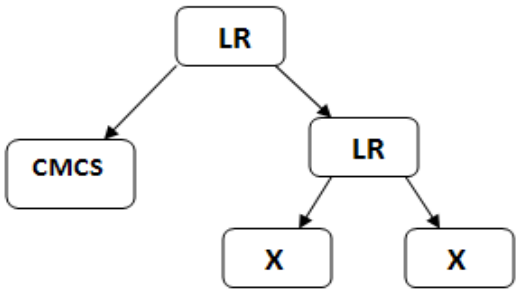
MAX12 – vraća veću između vrijednosti koje dobiva od podstabala.

FUNMIN – ako se program nalazi u dijelu stabla koji služi za izbacivanje stranice iz radnog spremnika, vraća indeks stranice za koju vrijedi da je *f(L, R)* minimalan, inače vraća vrijednost *f(L, R).* Na primjer, ako lijevo podstablo čvora vraća vrijednost tri, a desno dva, čvor će vratiti indeks stranice za koju vrijedi da je *f = 3*info1+2*info2* minimalan.

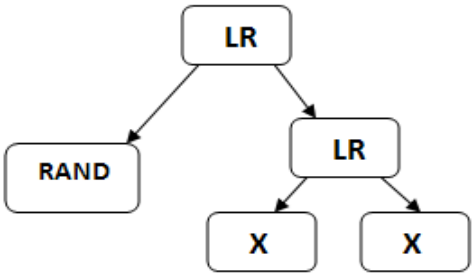
FUNMAX – sličan je čvoru *FUNMIN*, samo što traži maksimalnu vrijednost funkcije.

4.1.3 Prikaz osnovnih strategija izbacivanja stranica binarnim stablom

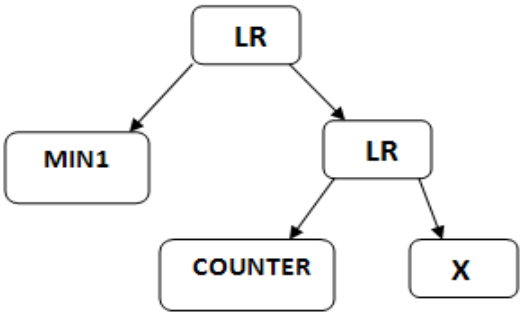
Pomoću navedenih čvorova je, između ostalog, moguće implementirati sve osnovne strategije izbacivanja stranica iz radnog spremnika. Na slici 4.1. dani su primjeri kako bi izgledala takva stabla.



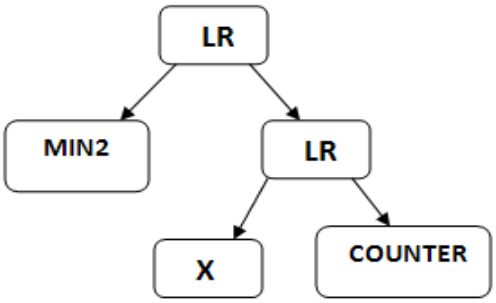
a) FIFO



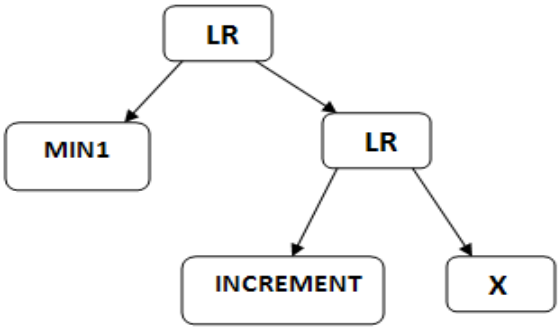
b) RAND



c) LRU (prvi način)



d) LRU (drugi način)



e) NFU

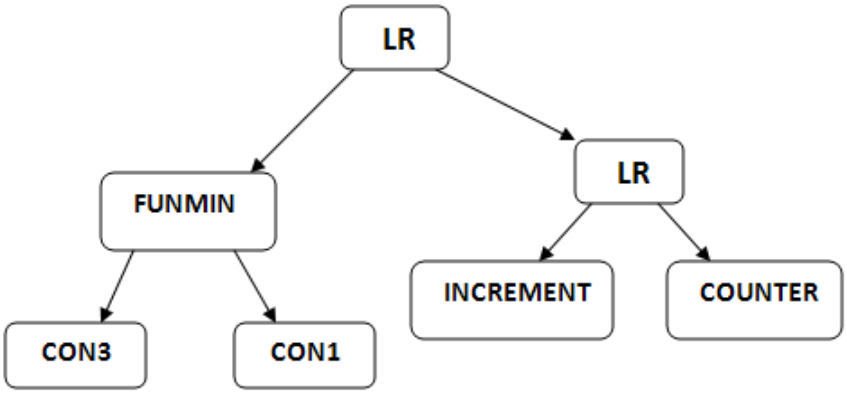
Slika 4.1. Reprezentacija osnovnih strategija izbacivanja stranica pomoću stabla

Pomoću ***LR*** čvorova (slika 4.1.) stablo je podijeljeno na tri dijela. Prvo podstablo (lijevo od početnog čvora) odabire stranicu koja će se izbaciti iz radnog spremnika, dok druga dva upisuju odgovarajuće vrijednosti u *info1* i *info2* referencirane stranice. *LR* čvorovi su ključni dio stabla, neophodan za ispravan rad jedinke (programa), te stoga moraju biti prisutni na odgovarajućim mjestima u svakom stablu (korijen stabla i njegovo desno dijete). Ostatak stabla se generira slučajno i ne sadrži *LR* čvorove.

Na slici 4.1. upotrebljen je čvor označen s ***X*** koji nije prethodno definiran. Taj čvor ne postoji u implementaciji samog genetskog programa, već označava da se na tom mjestu može nalaziti proizvoljan čvor ili podstablo (dotičan čvor/podstablo nema nikakvog utjecaja na izbor stranice koja će se izbaciti iz radnog spremnika). To je najbolje vidljivo kod strategija *FIFO* i *RANDOM* koje ne ovise o dodatnim informacijama, već izbacuju stranice *određenim redoslijedom* (*FIFO*), odnosno *slučajnim odabirom* (*RANDOM*).

Strategija *LRU* (i *LFU*) izbacuje stranice radnog spremnika ovisno o podacima koji su zapisani u poljima *info1* ili *info2* (izbacuje se stranica s minimalnom vrijednošću u polju informacija). U skladu s tim, moguće su dvije simetrične implementacije *LRU* algoritma (prikazane na slikama *4.1. c) i 4.1. d)*) ovisno o tome unutar kojeg polja informacija se traži stranica s minimalnom vrijednošću.

Osnovna zamisao upotrebe dva odvojena polja informacija (*info1* i *info2*) je u mogućnosti njihovog kombiniranja te donošenja odluke na temelju podataka iz oba polja. Primjer jednog takvog (još uvijek jednostavnog) stabla dan je na slici 4.2.



Slika 4.2. Primjer stabla koje koristi vrijednosti iz oba polja informacija

Jedinka zadana stablom prikazanim na slici 4.2. donosi odluku o izbacivanju stranice iz radnog spremnika na temelju vrijednosti u oba polja informacija. U polju *info1* se pamti broj referenciranja pojedine stranice u radnom spremniku, dok polje *info2* govori koje su stranice referencirane u bližoj prošlosti. Jedinka će odabrati stranicu koja će se izbaciti iz radnog spremnika na temelju sljedeće formule:

$$Index = min(CON3*info1 + CON1*info2) \qquad (4.1)$$

Interpretacija formule je sljedeća: za svaku se stranicu izračunava zbroj vrijednosti koja se nalazi u polju *info1* dotične stranice i vrijednosti u polju *info2* pomnoženom s tri, te se izbacuje stranica za koju je izračunata vrijednost minimalna.

Na slici 4.1. je vidljivo da implementacija osnovnih strategija izbacivanja stranica iz radnog spremnika pomoću binarnog stabla zahtijeva relativno malen broj čvorova, pa možemo pretpostaviti da će se dio tih strategija u postupku izvođenja genetskog programa doista i implementirati.

4.2 Osnovni parametri genetskog programa:

Za sam rad genetskog programa veoma je važno dobro odabrati *način računanja dobrote* jedinki, postupak *selekcije*, te načine *križanja* i *mutacije* jedinki. Navedeni parametri programa imaju veliku ulogu u procesu evolucije jedinki te će se stoga njihova kvaliteta odraziti i na kvalitetu konačnog rješenja.

Dobrota jedinki određuje koliko je koja jedinka „dobra“ te u kojoj je mjeri dotična jedinka bolja ili lošija od ostalih jedinki u populaciji. Kod genetskog programa za pronalaženje strategije zamjene stranica u radnom spremniku, kao mjera dobrote jedinke nameće se ukupan *broj pogodaka* ili ukupan *broj promašaja* koji je dotična jedinka ostvarila prilikom referenciranja niza stranica iz sekundarne memorije. Za računanje dobrote jedinke odabran je ukupan *broj promašaja* pa će bolje jedinke u ovom slučaju imati manju vrijednost dobrote (time nismo ograničili kvalitetu genetskog programa).

Kao postupak ***selekcije jedinki*** je odabran *turnirski eliminacijski odabir*:

- Slučajno se odabiru četiri jedinke iz populacije
- Eliminiraju se dvije najlošije

Turnirski eliminacijski odabir ima (važno) svojstvo *elitizma* – najbolja jedinka u generaciji (ili u ovom slučaju dvije najbolje) nikad ne će biti izbačena.

Načini ***križanja i mutacije jedinki*** jednaki su načinima opisanim u trećem poglavlju, ali moramo paziti da čvorovi *LR* ne sudjeluju u procesima križanja i mutacije (u suprotnom bismo mogli dobiti nemoguća rješenja).

Za rad genetskog programa potrebno je još definirati i ***uvjet zaustavljanja programa.*** To može biti unaprijed definirani broj generacija koje program mora ispitati, broj generacija bez poboljšanja ili zadana dobrota jedinki koju je potrebno postići. Kao uvjet zaustavljanja odabran je *unaprijed definirani broj generacija* koje program treba testirati (izvođenje programa je uvijek moguće i „nasilno“ prekinuti ako smatramo da je neka jedinka dovoljno dobra).

Osim gore navedenih parametara, koji su ključni za rad bilo kojeg genetskog programa, moguće je mijenjati i neke parametre koji su specifični za sam problem izbacivanja stranica iz radnog spremnika. To su, npr. *veličina radnog spremnika*, *veličina sekundarne memorije* ili *broj zahtjeva* za stranicama sekundarne memorije. Promjena tih parametara direktno utječe na kvalitetu pojedine jedinke, tj. na broj pogodaka ili promašaja koji je dotična jedinka ostvarila prilikom referenciranja niza stranica sekundarne memorije.

5 ANALIZA REZULTATA

Za potrebe ocjenjivanja genetskog programa provedeno je 28 pokusa iz kojih su preuzete najbolje jedinke. Ove jedinke služe za usporedbu s postojećim strategijama izbacivanja stranica iz radnog spremnika ili za provjeru kvalitete rada samog genetskog programa. Kvaliteta pojedinih jedinki određuje se usporedbom s *LRU (least recently used)*, *OPT (teoretski optimalna strategija)* i *NFU (not frequently used)* strategijama, dok se kvaliteta rada samog genetskog programa određuje na temelju *konstantnosti* (sličnosti u dobrotama najboljih dobivenih jedinki prilikom različitih pokretanja programa) i *kvalitete* dobivenih jedinki. Kvaliteta pojedine jedinke (ili strategije izbacivanja) izražena je omjerom *broja pogodaka* i ukupnog broja referenciranja stranica sekundarne memorije (u postocima).

Svaka je jedinka *generirana* na temelju jednog *primjera za učenje*. Primjeri za učenje sadrže *dvadeset tisuća* zahtjeva za stranicama sekundarne memorije, a razlikuju se po veličini radnog spremnika koji je korišten (veličina sekundarnog spremnika je za sve primjere jednaka).

Kvaliteta pojedine jedinke se određuje na temelju *skupa primjera za testiranje*. Primjeri za testiranje sadrže znatno veći broj zahtjeva te omogućuju realnu procjenu kvalitete dobivene jedinke.

5.1 Primjeri za učenje

Prilikom generiranja jedinki za usporedbu s postojećim strategijama izbacivanja, korištene su različite veličine radnog spremnika (*200, 250, 330* te *500* stranica) te su za svaku od navedenih veličina odabrane po dvije jedinke (najbolje jedinke iz zasebnih pokretanja genetskog programa). Pojedina jedinka se testira nad radnim spremnikom veličine jednake onom pomoću kojeg je generirana. U nastavku je dana usporedba kvalitete odabranih jedinki i osnovnih strategija izbacivanja stranica nad primjerima za učenje. Strategije dobivene genetskim programiranjem označene su s *GP1* i *GP2*.

Tablica 5.1. Usporedba strategija izbacivanja (omjer radne i sekundarne memorije 1/5)

1/5	OPT	LRU	NFU	GP1	GP2
Broj pogodaka	14.407	10.130	9.543	10.288	10.280
Postotak pogodaka	72.04%	50.65%	47.72%	51.44%	51.40%

Tablica 5.2. Usporedba strategija izbacivanja (omjer radne i sekundarne memorije 1/4)

1/4	OPT	LRU	NFU	GP1	GP2
Broj pogodaka	15.100	10.778	10.243	11.061	11.029
Postotak pogodaka	75.50%	53.89%	51.22%	55.30%	55.15%

Tablica 5.3. Usporedba strategija izbacivanja (omjer radne i sekundarne memorije 1/3)

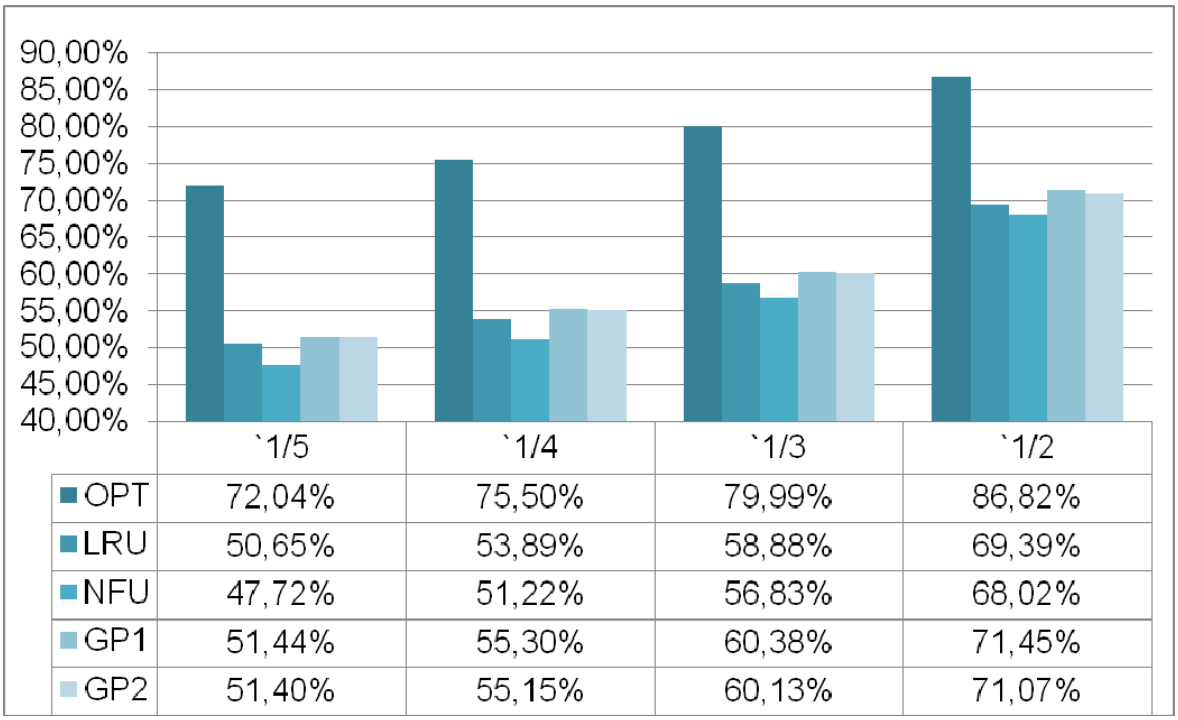
1/3	OPT	LRU	NFU	GP1	GP2
Broj pogodaka	15.998	11.775	11.366	12.076	12.025
Postotak pogodaka	79.99%	58.88%	56.83%	60.38%	60.13%

Tablica 5.4. Usporedba strategija izbacivanja (omjer radne i sekundarne memorije 1/2)

1/2	OPT	LRU	NFU	GP1	GP2
Broj pogodaka	17.363	13.878	13.604	14.290	14.213
Postotak pogodaka	86.82%	69.39%	68.02%	71.45%	71.07%

Iz predočenih tablica (te grafa 5.1.) vidljivo je da jedinke dobivene genetskim programiranjem *dominiraju* nad postojećim strategijama izbacivanja stranica (osim teoretski optimalne strategije koja nije izvediva u praksi). Takva pojava je i očekivana jer je genetski program kroz postupke selekcije, križanja i mutacije „prilagodio“ svoje jedinke kako bi bile što efikasnije za konkretan ulazni niz (*primjer za učenje jedinke*). Međutim, činjenica da je jedinka „dobra“ za neki ulazni niz, ne garantira da će dotična jedinka općenito predstavljati kvalitetnu strategiju izbacivanja stranica.

Graf 5.1. usporedba strategija izbacivanja nad primjerima za učenje



	`1/5	`1/4	`1/3	`1/2
OPT	72,04%	75,50%	79,99%	86,82%
LRU	50,65%	53,89%	58,88%	69,39%
NFU	47,72%	51,22%	56,83%	68,02%
GP1	51,44%	55,30%	60,38%	71,45%
GP2	51,40%	55,15%	60,13%	71,07%

5.2 Primjeri za testiranje

Primjeri za testiranje sadrže znatno veći broj zahtjeva od primjera za učenje te služe za stvarnu procjenu kvalitete pojedinih jedinki. Kao informacija o kvaliteti koristi se *prosječan postotak pogodaka* koji je jedinka ostvarila (nad primjerima za testiranje). Jedinke se testiraju uz radni spremnik veličine jednake onom pomoću kojeg su generirane.

Tablica 5.5. Usporedba strategija izbacivanja (primjeri za testiranje)

1/5	OPT	LRU	NFU	GP1	GP2
Postotak pogodaka	73.06%	52.08%	51.39%	52.12%	52.04%

Tablica 5.6. Usporedba strategija izbacivanja (primjeri za testiranje)

1/4	OPT	LRU	NFU	GP1	GP2
Postotak pogodaka	76.73%	55.15%	54.43%	55.06%	55.08%

Tablica 5.7. Usporedba strategija izbacivanja (primjeri za testiranje)

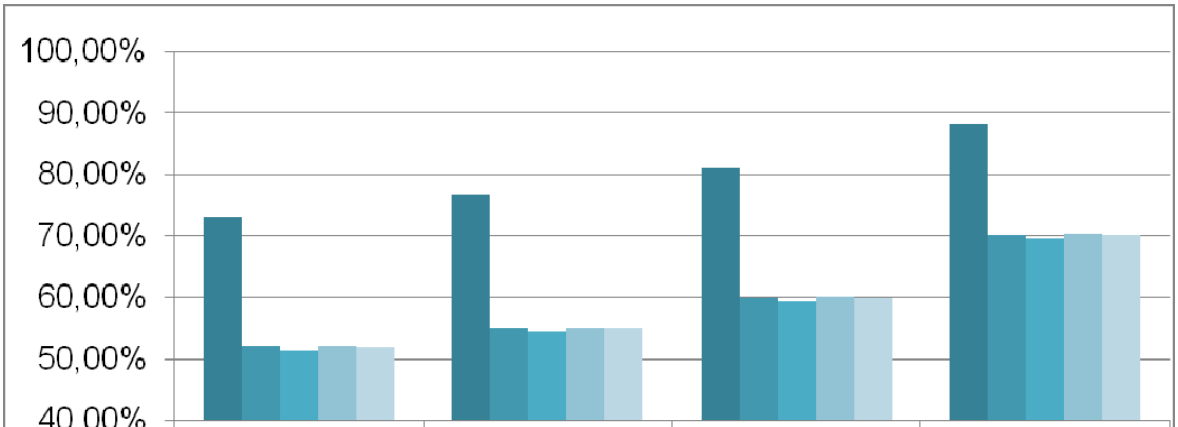
1/3	OPT	LRU	NFU	GP1	GP2
Postotak pogodaka	81.10%	60.01%	59.27%	60.10%	60.02%

Tablica 5.8. Usporedba strategija izbacivanja (primjeri za testiranje)

1/2	OPT	LRU	NFU	GP1	GP2
Postotak pogodaka	88.34%	70.29%	69.65%	70.34%	70.22%

U tablicama 5.5. do 5.8. prikazane su kvalitete pojedinih strategija izbacivanja stranica iz radnog spremnika (uključujući i dvije genetskim programom generirane jedinke). Vidljivo je da generirane jedinke dominiraju nad NFU algoritmom za sve testirane omjere radne i sekundarne memorije, dok se pri usporedbi s LRU algoritmom dobivaju približno jednaki rezultati.

Graf 5.2. usporedba strategija izbacivanja nad primjerima za testiranje

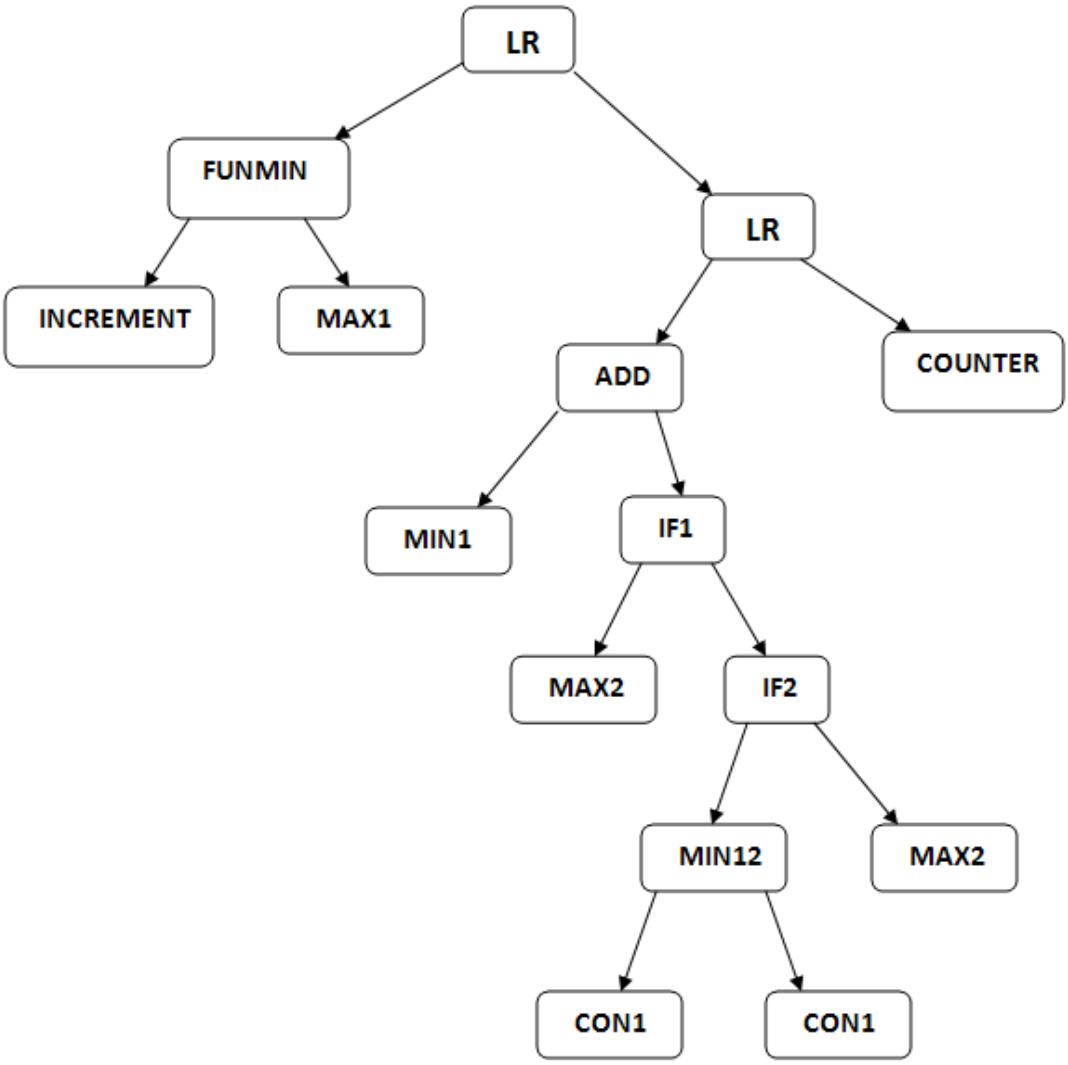


	`1/5	`1/4	`1/3	`1/2
OPT	73,06%	76,73%	81,10%	88,34%
LRU	52,08%	55,15%	60,01%	70,29%
NFU	51,39%	54,43%	59,27%	69,65%
GP1	52,12%	55,06%	60,10%	70,34%
GP2	52,04%	55,08%	60,02%	70,22%

5.3 Kvaliteta rada genetskog programa

Kvaliteta rada genetskog programa određuje se na temelju *konstantnosti* (sličnosti u dobrotama najboljih dobivenih jedinki prilikom različitih pokretanja programa) i *kvalitete* dobivenih jedinki. Kvaliteta dobivenih jedinki pokazana je u prethodnom poglavlju pa je još potrebno pokazati koliko često genetski program generira kvalitetne jedinke te u kojoj mjeri.

Za potrebe određivanja kvalitete rada genetskog programa generirano je dvadeset jedinki uz omjer veličine radne i sekundarne memorije jedan naprema četiri (broj zahtjeva za stranicama iznosi *dvadeset tisuća*). *Srednja vrijednost* broja *pogodaka* generiranih jedinki iznosi *10.976* uz *standardnu devijaciju* od *34,08*. Dobiveni rezultati pokazuju da genetski program generira podjednako dobre jedinke za istovjetne početne parametre. Najbolja od dobivenih jedinki (sa 11.061 pogodaka) prikazana je na slici 5.1.



Slika 5.1. Prikaz jedinke generirane genetskim programom

6 ZAKLJUČAK

Genetskim programiranjem dobivene su jedinke (algoritmi izbacivanja stranica) koje su podjednako kvalitetne kao *LRU* strategija, dok nad ostalim osnovnim strategijama izbacivanja stranica *dominiraju* (*NRU, FIFO, RAND*). Kvaliteta dobivenih strategija (prilikom provjere jedinki nad *primjerima za testiranje)* pokazuje da su *primjeri za učenje* dovoljno reprezentativni te će generirane jedinke davati zadovoljavajuće rezultate prilikom testiranja različitih nizova zahtjeva za stranicama sekundarne memorije.

Osnovni problem generiranih strategija je njihova *složenost izvedbe* u računalu, te *vremenski i memorijski resursi* koje bi takve strategije zauzimale. Dva polja informacija (*info1* i *info2*) proširuju mogućnosti generiranih strategija, ali zauzimaju dvorstruko više memorije (od npr. *LRU* strategije koja ima samo jedno polje informacije – vrijeme zadnjeg referenciranja stranice). Osim toga, prilikom svakog zahtjeva za stranicom sekundarne memorije, potrebno je upisati broj u svako od polja informacija, što udvostručuje vrijeme potrebno za određivanje stranice koja će se izbaciti iz radne memorije.

Dodatan problem predstavlja *složenost binarnog stabla* koje reprezentira strategiju izbacivanja stranica (koje se često sastoji od više desetaka čvorova). Vrijeme potrebno za određivanje indeksa stranice koja će se izbaciti iz radnog spremnika te računanje informacija koje će se upisati u polja *info1* i *info2* time se dodatno produžuje. Taj se problem ne manifestira ako je stablo ugrađeno u tekst programa, a ne kao prilikom učenja, gdje se mora prikazati dinamički.

Genetskim programiranjem moguće je dobiti strategije izbacivanja stranica iz radnog spremnika koje su jednako dobre ili bolje od postojećih, ali zbog svoje složenosti te nedostatka sklopovske podrške nisu prikladne za implementaciju u računalima.

7 LITERATURA

Jakobović, D. Genetski algoritmi – predavanje, 2007.

Megiddo, N., Modha, D. S., *Outperforming LRU with an adaptive replacement cache algorithm*, Computer, Volume 37, Issue 4, April 2004. Page(s): 58 - 65

Ribarić, S. Arhitektura računala RICS i CISC. Zagreb: Školska knjiga d. d., 1996.

Cache, http://en.wikipedia.org/wiki/Cache, datum pristupa: 10.06.2008.

8 SAŽETAK

U radu je opisano ostvarenje *strategije zamjene stranica* u radnom spremniku uz pomoć genetskog programiranja. Dan je pregled postojećih strategija zamjene stranica te njihove osnovne prednosti i nedostaci. U nastavku je opisan princip rada i osnovni pojmovi vezani uz genetsko programiranje (prikaz jedinki, generiranje početne populacije, računanje dobrote jedinki te postupci selekcije, križanja i mutacije).

Svaka jedinka genetskog programa predstavlja jednu strategiju zamjene stranica, prikazanu u obliku binarnog stabla. Prilikom generiranja pojedinih jedinki (strategija) korišteni su *primjeri za učenje,* dok se kvaliteta pojedine strategije određuje nad *primjerima za testiranje jedinki*, usporedbom s postojećim algoritmima zamjene stranica (*LRU, NRU, OPT*). Dobiveni rezultati iskorišteni su za procjenu kvalitete genetskog programa te procjenu isplativosti implementacije dobivenih strategija u računalnim sustavima.

Ključne riječi: *strategije zamjene stranica, genetsko programiranje, primjeri za učenje, primjeri za testiranje*

9 ***ABSTRACT***

Title: Obtaining primary memory page replacement strategy with genetic programming

This paper describes the implementation of page replacement algorithms by using genetic programming. The strategy rewiev for page replacement algorithms and their basic advantages and disadvantages are described. The following describes the principle and basic notions related to genetic programming (individual representation, generating initial population, computing fitness and methods for selection, crossing and mutations).

Each individual of a genetic program represents a strategy for page replacement, represented in the form of a binary tree. When generating individuals (strategies), learning samples were used, while the quality of individual strategy is determined by the testing samples, compared with the existing page replacement algorithms (LRU, NRU, OPT). The obtained results provide an estimate for the genetic program quality, and feasibility assesment of implementation in computer systems.

Key words: page replacement algorithms, genetic programming, learning samples, testing samples